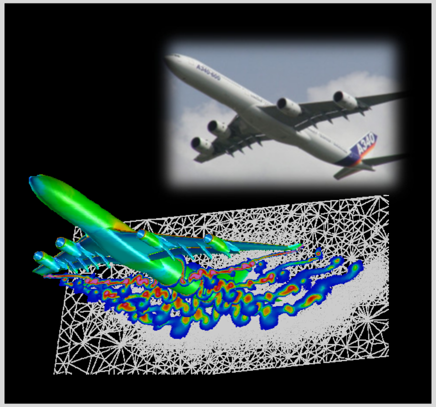


High Performance Computing

ADVANCED SCIENTIFIC COMPUTING

Prof. Dr. – Ing. Morris Riedel

Adjunct Associated Professor

School of Engineering and Natural Sciences, University of Iceland, Reykjavik, Iceland

Research Group Leader, Juelich Supercomputing Centre, Forschungszentrum Juelich, Germany

@Morris Riedel @MorrisRiedel @MorrisRiedel

LECTURE 4

Advanced MPI Techniques

September 19, 2019

Room V02-258

UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

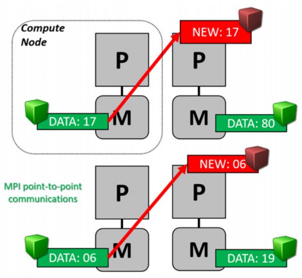
DEEP Projects

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

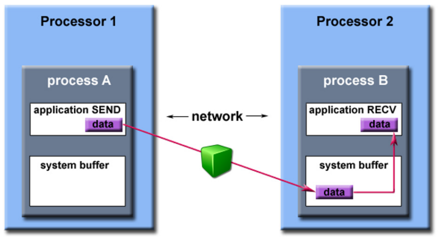
HAICU HELMHOLTZ ARTIFICIAL INTELLIGENCE COOPERATION UNIT

Review of Practical Lecture 3.1 – Understanding MPI Messages & Collectives

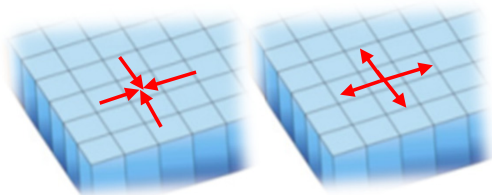
- MPI purpose: send data as messages to other processors



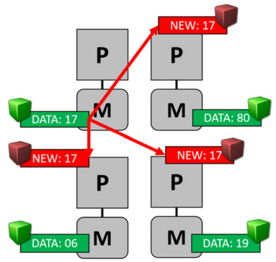
MPI
Point
to
Point
Communication

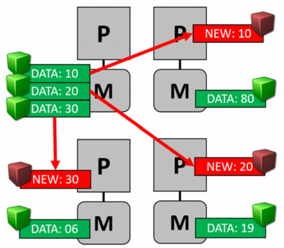



```
MPI_Init(&argc, &argv);

MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

if (rank == 0) {
  dest = 1; source = 1;
  rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
}
else if (rank ==1 ) {
  dest = 0; source = 0;
  rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
  rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
}

rc = MPI_Get_count(&Stat, MPI_CHAR, &count);
```



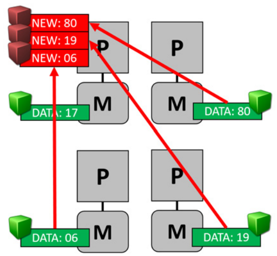
(e.g. using MPI messages in scientific simulations)

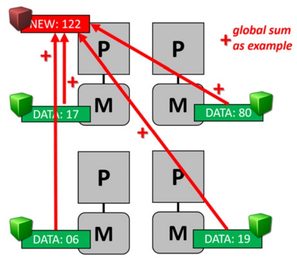




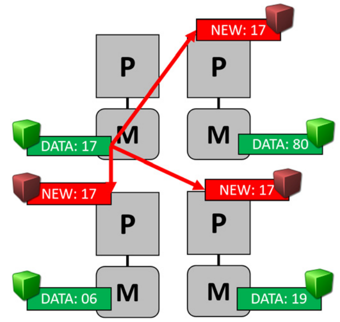
MPI
Collective
Communication

[1] LLNL MPI Tutorial






```
MPI_Init(&argc,&argv);

MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);

source=0;
count=4;

if(rank == source){
  for(i=0;i<count;i++)
    buffer[i]=i;
}

MPI_Bcast(buffer,count,MPI_INT,source,MPI_COMM_WORLD);

for(i=0;i<count;i++)
  printf("%d \n",buffer[i]);

MPI_Finalize();
```



(e.g. instead of using a for loop & MPI_Send/Recv multiple times)

Outline of the Course

1. High Performance Computing
2. Parallel Programming with MPI
3. Parallelization Fundamentals
4. Advanced MPI Techniques
5. Parallel Algorithms & Data Structures
6. Parallel Programming with OpenMP
7. Graphical Processing Units (GPUs)
8. Parallel & Scalable Machine & Deep Learning
9. Debugging & Profiling & Performance Toolsets
10. Hybrid Programming & Patterns
11. Scientific Visualization & Scalable Infrastructures
12. Terrestrial Systems & Climate
13. Systems Biology & Bioinformatics
14. Molecular Systems & Libraries
15. Computational Fluid Dynamics & Finite Elements
16. Epilogue

+ additional practical lectures & Webinars for our hands-on assignments in context

- Practical Topics
- Theoretical / Conceptual Topics

Outline

- Advanced MPI Communication Techniques
  - Blocking vs. Non-Blocking Communication
  - MPI Communicators & Creating Sub-Groups
  - MPI Cartesian Communicator & Application Motivations
  - Hardware & Communication Issues & Network Interconnects
  - Task-Core Mappings & Heatmap Application Example

- MPI Parallel I/O Techniques
  - I/O Terminologies & Challenges
  - Parallel Filesystems & Striping Technique
  - MPI I/O Techniques & Use of Parallel I/O
  - Higher-Level I/O Libraries & Community Standards
  - Portable File Formats & Data Science Application Example

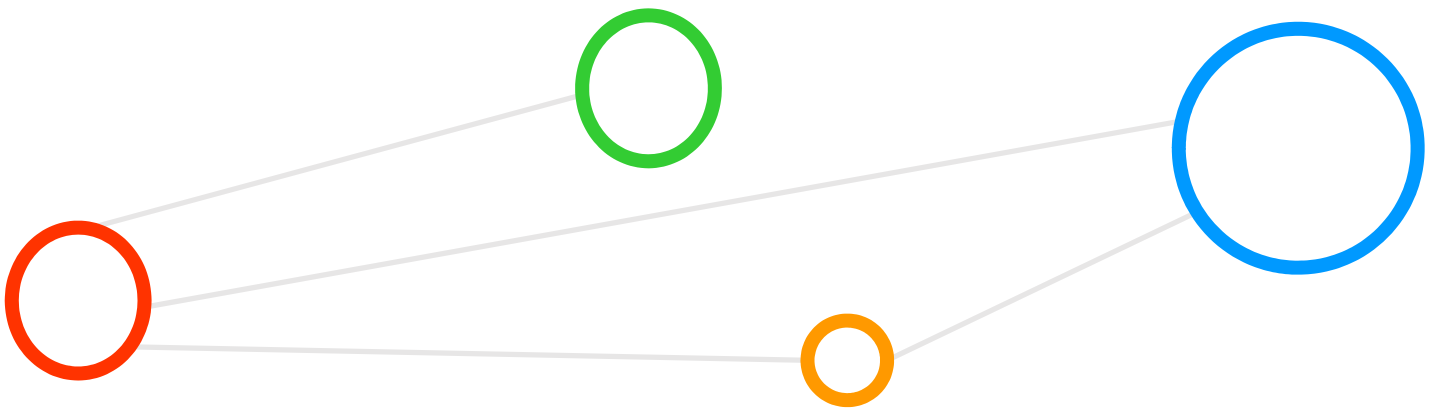
- **Promises from previous lecture(s):**
- ***Lecture 1:*** **Lecture 2 & 4 will give in-depth details on the distributed-memory programming model with the Message Passing Interface (MPI)**
- ***Lecture 2:*** **Lecture 4 will provide more details on advanced functions of the Message Passing Interface (MPI) standard & its use in applications**
- ***Lecture 2:*** **Lecture 4 on advanded MPI techniques will provide details about the often used MPI cartesian communicator & its use in applications**
- ***Practical Lecture 3.1:*** **Lecture 4 will offer more insights about using different types of MPI communicators with different rank identities in MPI applications**
- ***Practical Lecture 3.1:*** **Lecture 4 will offer more insights about using blocking communication vs. non-blocking communication functions when using MPI**
- ***Practical Lecture 3.1:*** **Lecture 4 will offer more insights about using the MPI status for different purposes and to obtain a better understanding what happens**

Selected Learning Outcomes

- Students understand…
  - Latest developments in parallel processing & high performance computing (HPC)
  - How to create and use high-performance clusters
  - What are scalable networks & data-intensive workloads
  - The importance of domain decomposition
  - Complex aspects of parallel programming
  - HPC environment tools that support programming or analyze behaviour
  - Different abstractions of parallel computing on various levels
  - Foundations and approaches of scientific domain-specific applications

- Students are able to …
  - Programm and use HPC programming paradigms
  - Take advantage of innovative scientific computing simulations & technology
  - Work with technologies and tools to handle parallelism complexity

Advanced MPI Communication Techniques

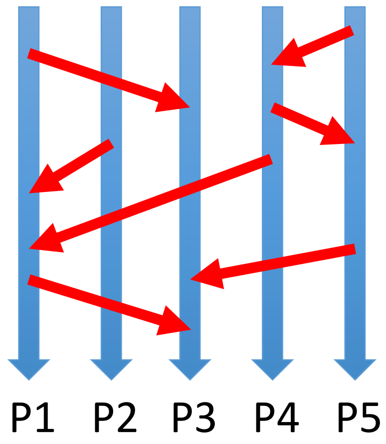


Programming with Distributed Memory using MPI – Revisited (cf. Lecture 1)

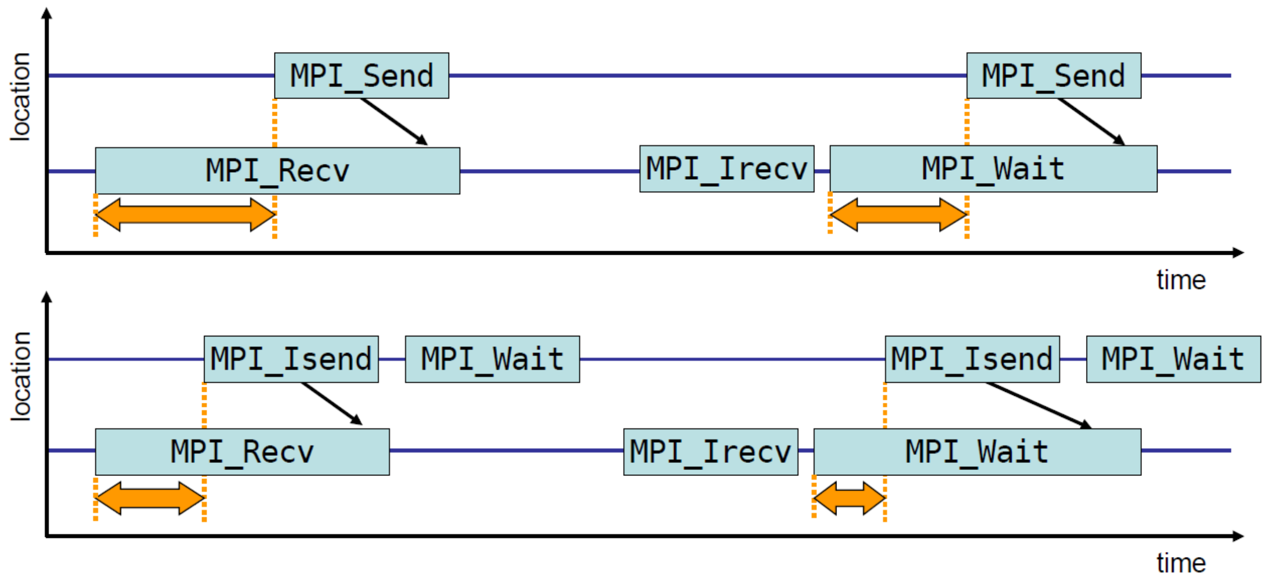
- **Distributed-memory programming enables explicit message passing as communication between processors**
- **Message Passing Interface (MPI) is dominant distributed-memory programming standard today (available in many different version)**
- **MPI is a standard defined and developed by the MPI Forum**

[4] MPI Standard

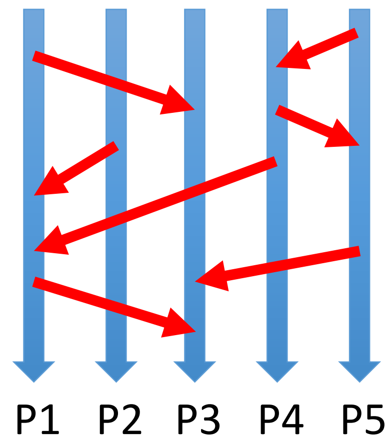
- Features
  - No remote memory access on distributed-memory systems
  - Require to ‘send messages’ back and forth between processes PX
  - Many free Message Passing Interface (MPI) libraries available
  - Programming is tedious & complicated, but most flexible method



Blocking vs. Non-blocking communication



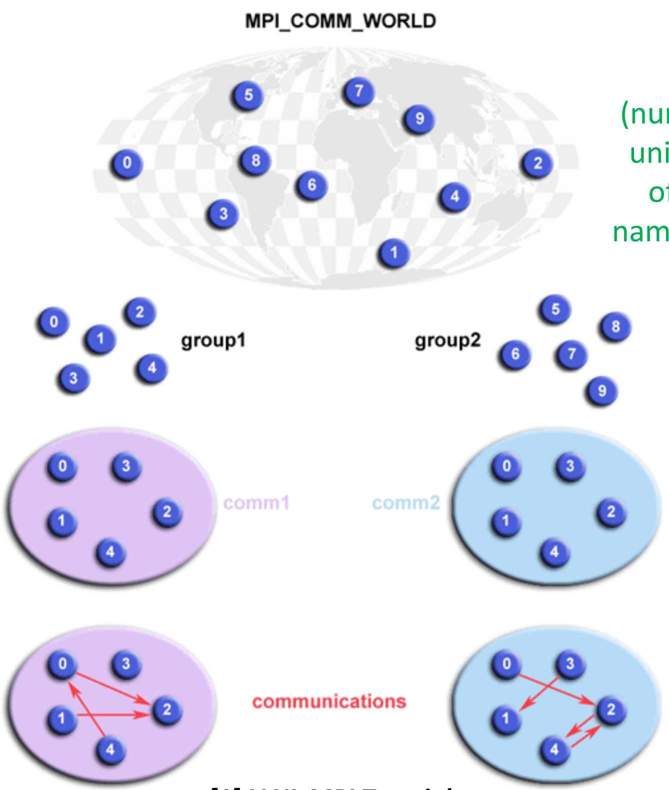
[5] Metrics tour



- **Blocking vs. non-blocking: MPI_Send() blocks until data is received; MPI_Isend() continues**
- **The use of these functions can cause different performance problems (e.g. here 'late sender')**
- **MPI_Wait() does wait for a given MPI request to complete before continuing**
- **MPI_Waitall() does wait for all given MPI requests (e.g. waiting for message) to complete before continuing**

➢ **Lecture 5 offers more details on using blocking & non-blocking MPI communication in simulations and data science applications**

Using MPI Ranks & Communicators – Revisited (cf. Lecture 2)



(numbers reflect unique identity of processor named 'MPI rank)

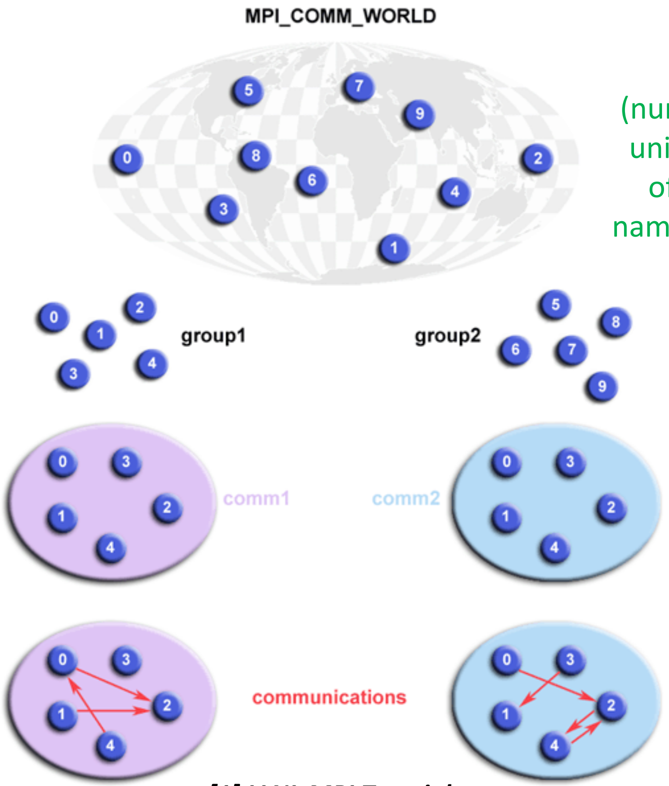
[1] LLNL MPI Tutorial

- Answers the following question:
  - How do I know where to send/receive to/from?
- Each MPI activity specifies the context in which a corresponding function is performed
  - MPI_COMM_WORLD (region/context of all processes)
  - Create (sub-)groups of the processes / virtual groups of processes
  - Peform communications only within these sub-groups easily with well-defined processes

- **Using communicators wisely in collective functions can reduce the number of affected processors**
- **MPI rank is a unique number for each processor**

MPI Communicators – MPI Create Sub-Group Communicators

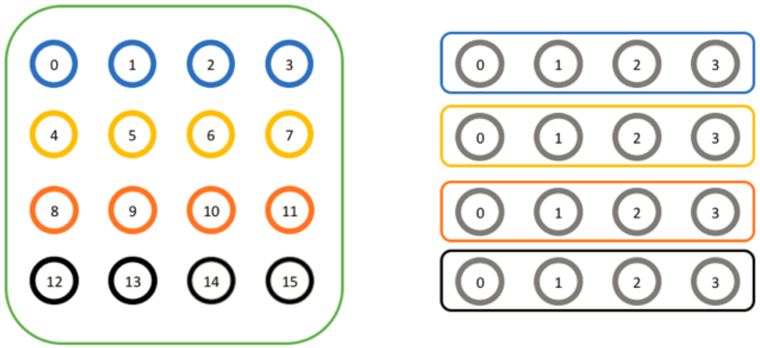
- Create (sub-)groups of the processes & virtual groups of processes
  - Simple to complex communicator setups
  - E.g. split existing communicator using `MPI_Comm_split()`
  - Free new communictors: `MPI_Comm_free()`



[1] LLNL MPI Tutorial

(numbers reflect unique identity of processor named 'MPI rank)

- **MPI_Comm_split() creates a new smaller communicator out of a larger communicator by splitting its current ranks (e.g., rows)**
- **New rank identities are created in the newly created communicator**
- **MPI ranks in different communicators represent different unique identifiers**



[2] Introduction to Groups & Communicators

Using MPI_Comm_split() & MPI_Comm_free() – Row Communicator Example

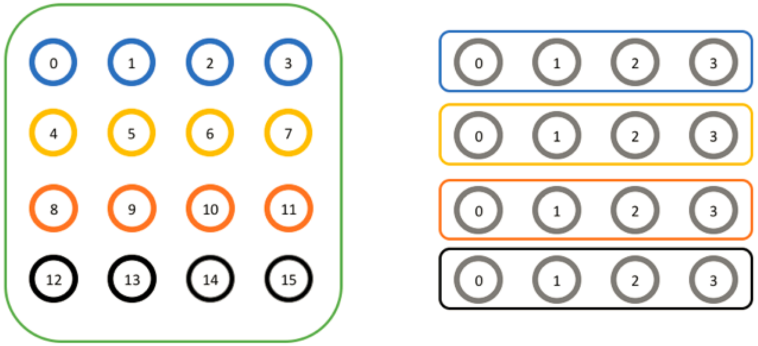
```
#include <stdio.h>
#include <mpi.h>
int main (int argc, char** argv) {
  int world_rank, world_size;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int color = world_rank / 4;
  MPI_Comm row_comm;
  MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &row_comm);
  int row_rank, row_size;
  MPI_Comm_size(row_comm, &row_size);
  MPI_Comm_rank(row_comm, &row_rank);
  printf("WORLD RANK/SIZE: %d/%d \t ROW RANK/SIZE: %d/%d\n",
         world_rank, world_size, row_rank, row_size);
  MPI_Comm_free(&row_comm);
  MPI_Finalize();
  return 0;
}
```

- **MPI_COMM_WORLD with all processors (cf. Lecture 2)**
- **Splitting scheme according to illustration matching colors / rows**

- **Definition of a new communicator and a split of the existing MPI_COMM_WORLD communicator using the defined row scheme via the function MPI_Comm_split()**

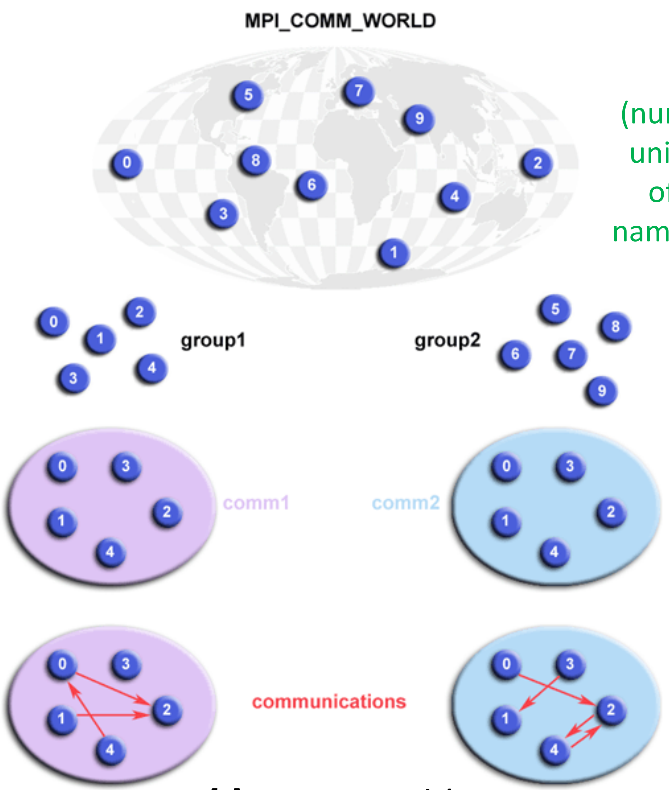
- **Different ranks and sizes for the newly created row communicator**
- **Print different identities in both communicators shows differences**

- **Free communicator**



[2] Introduction to Groups & Communicators

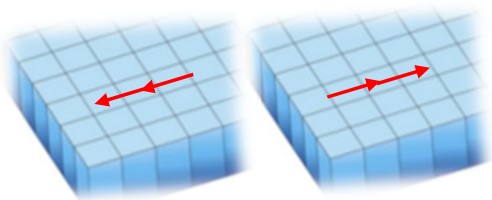
MPI Communicators – Create MPI Cartesian Communicators



[1] LLNL MPI Tutorial

(numbers reflect unique identity of processor named 'MPI rank)

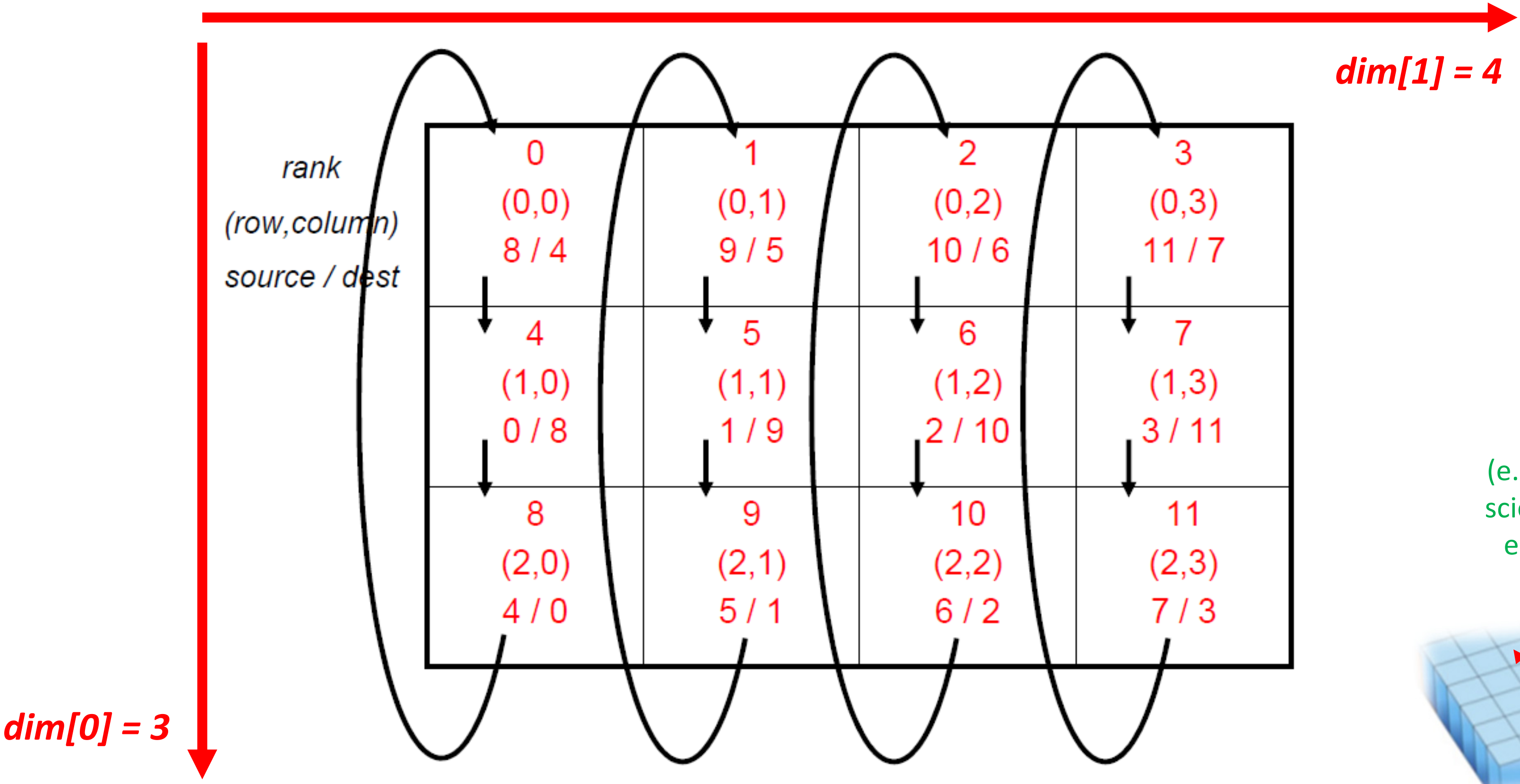
- Create (sub-)groups of the processes & virtual groups of processes
  - E.g. optimized for cartesian topology `MPI_Cart_create()`
  - Creates a new communicator out of MPI_COMM_WORLD
  - Dims: array with length for each dimension
  - Periods: logical array specifying whether the grid is periodic or not
  - Reorder: Allow reordering of ranks in output communicator



(e.g. using MPI messages in scientific simulations and/or engineering applications)

> ➢ **Assignment #3 will make use of the cartesian communicator in a simple application example that includes the moving of boats & fish**

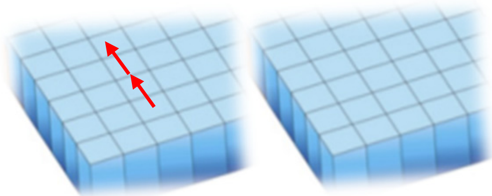
Cartesian Communicator Example – Conceptual View



('cartesian structure')

modified from [3] German MPI Lecture

(e.g. using MPI messages in scientific simulations and/or engineering applications)



Cartesian Communicator Example – Source-code View

```
#include <stdio.h>
#include <mpi.h>
int main (int argc, char** argv) {
  int rank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  dims[0]=3; dims[1] = 4;
  periods[0]=true; periods[1]=true;
  reorder = false;
  MPI_Cart_create(MPI_COMM_WORLD, 2, dims,
            periods, reorder, &comm_2d);
  MPI_Cart_coords(comm_2d, rank, 2, &coords);
  MPI_Cart_shift(comm_2d, 0, 1, &source, &dest);
  a = rank; b = 1;
  MPI_Sendrecv(a, 1, MPI_REAL, dest, 13, b, 1,
            MPI_REAL, source, 13, comm_2d, &status);
  MPI_Finalize();
  return 0;
}
```

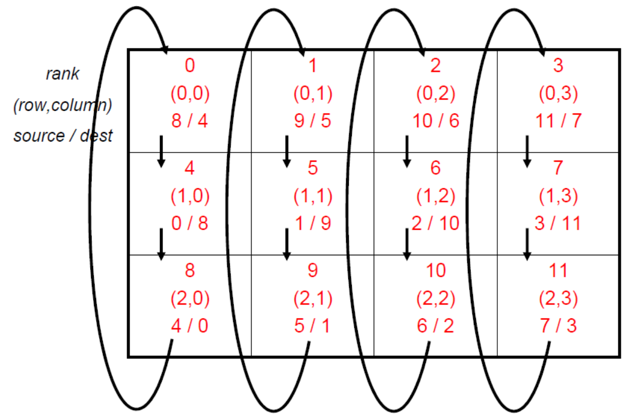
- **Preparing parameter dims as array with length for each dimension (here 3 x 4)**
- **Preparing parameter periods as logical array specifying whether the cartesian grid is period**
- **Preparing parameter reorder as not reordering of ranks in output communicator**

- **MPI_Cart_create() creates a new communicator (cartesian structure) according to specified dimensions in variables**

- **MPI_Cart_coords() obtains process coordinates in cartesian topology – note that this JUST obtaines the current process coordinates – no actual shift is done yet**
- **MPI_Cart_shift() obtains 'ranks' for shifting data in cartesian topology – note that this JUST prepares for a shift understanding which ranks are affected by shift**

- **A real shift is done using a typical MPI message exchange with the obtained ranks and in the space of the Cartesian communicator**

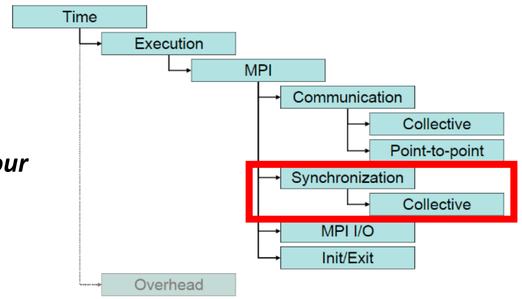
modified from [3] German MPI Lecture



Hardware & Communication Issues

- **Communication overhead can have significant impact on application performance**
- **Characteristics of interconnects of compute nodes/cpus affect parallel performance**

- Parallel Programming can cause communication issues
  - E.g. need for synchronisation in applications, e.g use of `MPI_Barriers()`
- Wide varieties of networks in HPC systems are available
  - Different network topologies of different types of networks used in HPC
- Gigabit Ethernet
  - Simple/cheep and good for high throughput computing (HTC)
  - Often too slow for parallel programs that require fast interconnects
- Infiniband
  - Fast, thus dominant distributed-memory computing interconnect today
  - Other interconnects exist but still less used: Intel Omnipath, Extoll, etc.



[5] Metrics tour

Communication Issues – Synchronisation with MPI Barrier Example

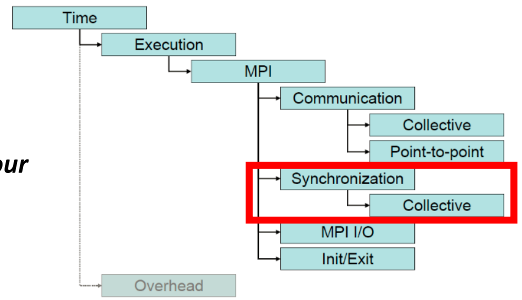
```
#include <stdio.h>
#include <mpi.h>
#include <unistd.h>
int main (int argc, char** argv) {
  int rank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if (rank == 0) {
    sleep(10);
  }
  if (rank == 1) {
    storeResultsToFile();
  }
  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Finalize();
  return 0;
}
```

- **One reason to require a synchronisation across processors is that one rank is performing some extraordinary long work, not others (e.g., master/worker parallelization technique, cf. Lecture 2)**
- **Sleep() is a function that puts a processor to sleep and thus doing basically nothing. Still a parallel computing resource is not usable for other users since it is typically exclusively allocated to one user by a scheduling system**

- **Another reason to require a synchronisation across processors is that one rank performs I/O operations of some kind (e.g., later in this lecture w/o using parallel I/O)**

- **MPI_Barrier() blocks the caller until all processes in the communicator have called it for synchronisation**

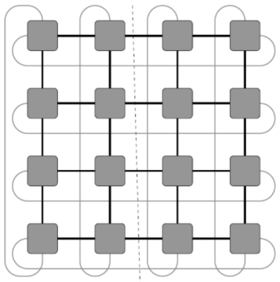
[5] Metrics tour

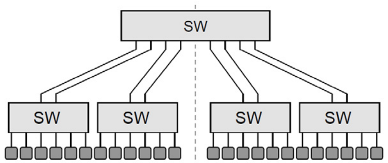


➢ **Lecture 9 on debugging, profiling & performance toolsets offers insights into performance analysis tools to understand MPI code better**

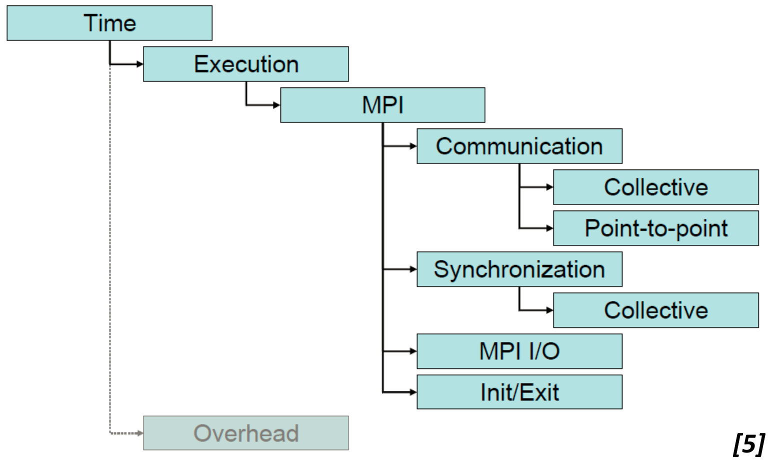
Optimization & Dependencies on Hardware & I/O

- Optimizations in terms of software & hardware are important
  - Optimization can be interpreted as using ‘dedicated‘ hardware features (if available)
  - E.g. network interconnections enable different used ‘network topologies‘ (varies in different systems)
  - E.g. parallel codes are tuned applying parallel I/O with parallel filesystems (if parallel filesystem exists)





[6] Introduction to High Performance Computing for Scientists and Engineers

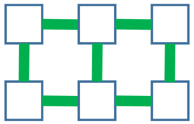


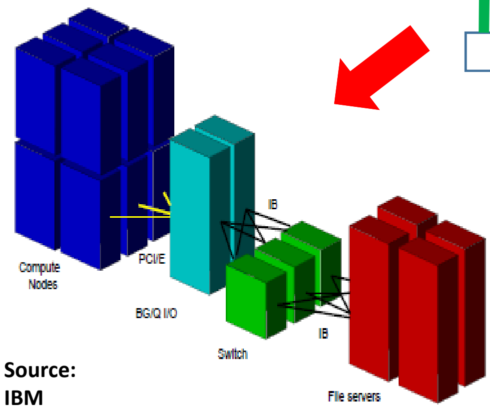
[5] Metrics tour

> **Lecture 9 on debugging, profiling & performance toolsets offers insights into performance analysis tools to understand MPI code better**

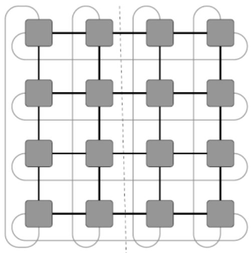
Complex Network Topologies & Challenges

- Large-scale HPC Systems have special network setups
  - Dedicated I/O nodes, fast interconnects, e.g. Infiniband (IB), Extoll, etc.
  - Different network topologies, e.g. tree, 5D Torus network, mesh, etc.
    (raise challenges in task mappings and communication patterns)

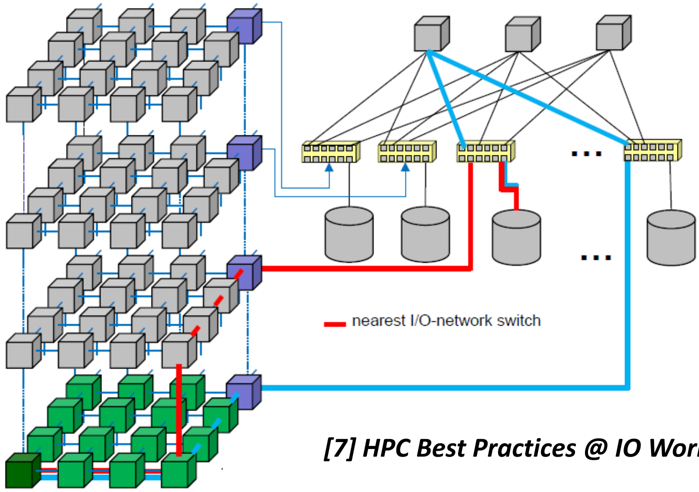




Source:
IBM



Contention and cross traffic caused by multiple I/O streams

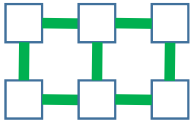


[6] Introduction to High Performance Computing for Scientists and Engineers

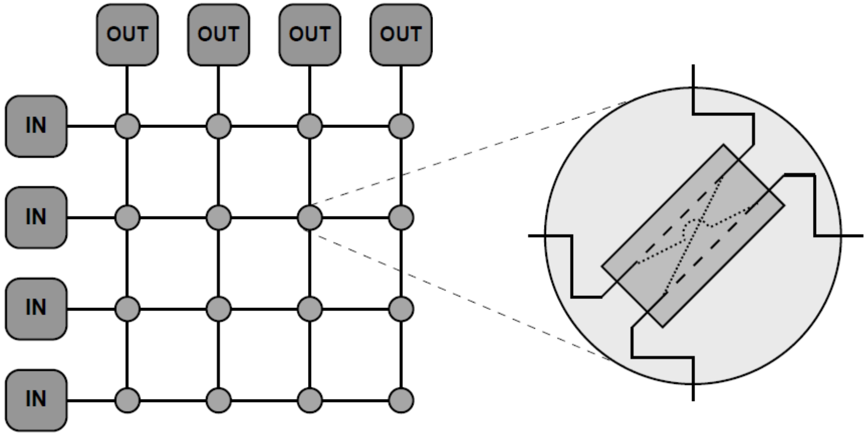
[7] HPC Best Practices @ IO Workshop

Network Building Block 'Switch' inside a HPC system

- A switch is an important network building block inside a HPC system that affects performance
- Think about workers processing data and interacting with each other → switch matters!
- Advanced programming techniques need to take the hardware interconnect into account



- Single fully non-blocking switch
  - All pairs of ports can use their full bandwidth concurrently
  - E.g. 2D cross-bar switch and each circle represents possible connections between two involved IN/OUT devices
  - Aka '2x2 switching element'
  - Aka 'four-port non-blocking switch'

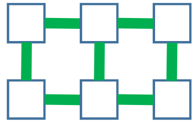


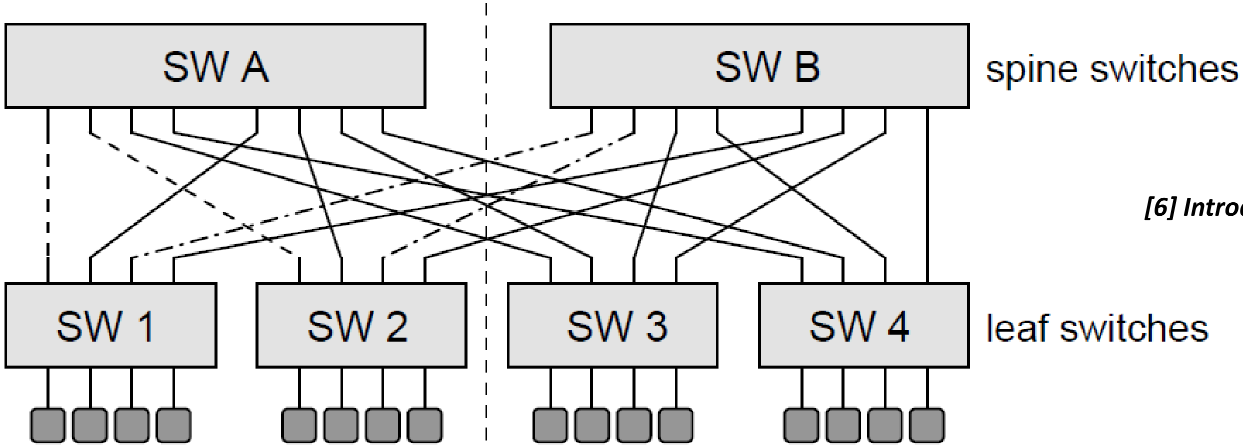
[6] Introduction to High Performance Computing for Scientists and Engineers

- Alternative
  - [partly/completely] single switch with bus design with limited bandwidth

Combining Network Building Blocks as FatTree (1)

- Fully non-blocking full-bandwidth fat-tree network
  - Having two switch layers (leaf and spine)
  - Keeps the 'non-blocking' feature across the whole system via two layers



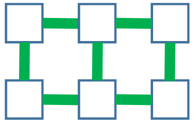


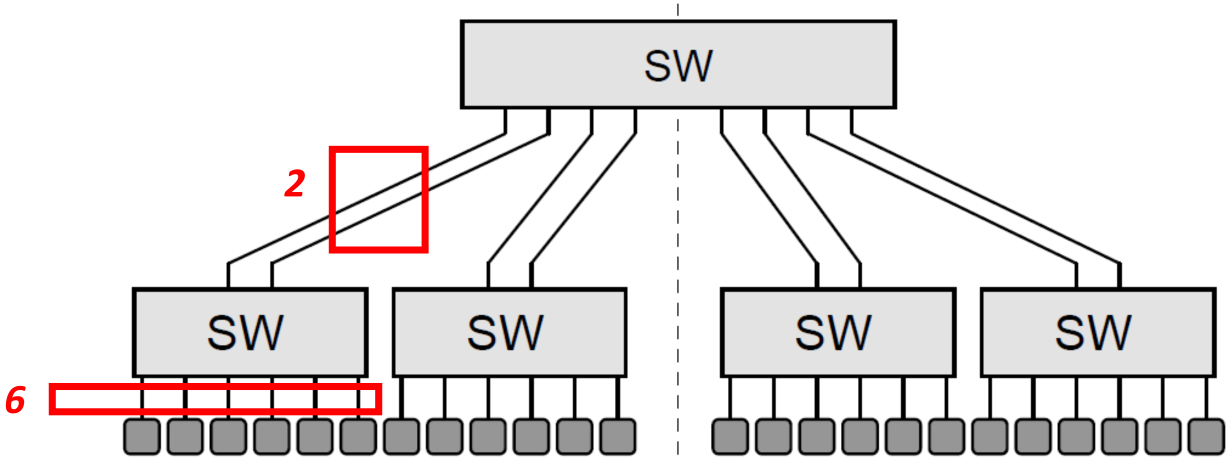
[6] Introduction to High Performance Computing for Scientists and Engineers

- **Here a group of workers processing data 'enjoy' full non-blocking communication**
- **Location of the workers here is not very crucial to the application performance**

Combining Network Building Blocks as FatTree (2)

- Fat-tree network with bottleneck (when # CPUs high)
  - Bottleneck is '1:3 oversubscription' of communication link to spine
  - Only four nonblocking pairs of connections are possible
  - Common in very large systems → safe costs (cable & switch hardware)

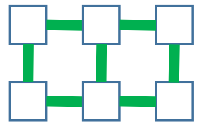




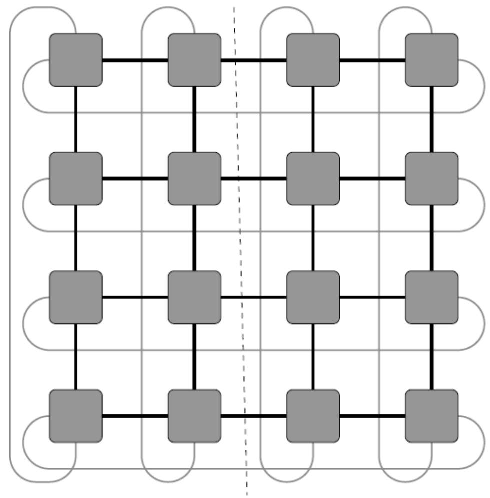
[6] Introduction to High Performance Computing for Scientists and Engineers

- **The location of the workers processing data is crucial for application performance here**

Mesh Networks



- Selected Facts
  - Often in the form of multi-dimensional hypercubes
  - Computing entity is located at each Cartesian grid intersection
  - Idea: connections are wrapped around the boundaries of the hypercube to form a certain torus topology
  - No direct connections between entities that are not next neighbours (but ok!)



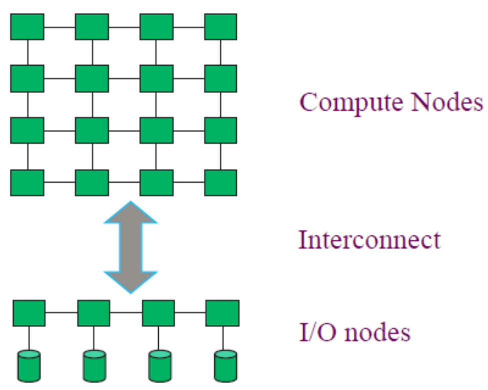
- Example: A 2D torus network
  - Bisection bandwidth scales with $\sqrt{N}$

[6] Introduction to High Performance Computing for Scientists and Engineers

- **Fat-Tree switches have limited scalability in very large systems (price vs. performance)**
- **Bisection bandwidth with scaling in large systems often via mesh networks (e.g. 2D torus)**

Example of Large-scale HPC Machine & I/O Setup

- Example: JUQUEEN
  - IBM BlueGene/Q
- Compute Nodes
  - 28 racks (7 rows à 4 racks)
    28,672 nodes (458,752 cores)
  - Rack: 2 midplanes à
    16 nodeboards (16,384 cores)
  - Nodeboard: 32 compute nodes
  - Node: 16 cores
- Dedicated I/O Nodes
  - 248 (27x8 + 1x32) connected to 2 CISCO Switches



[9] R. Thakur, PRACE Training, Parallel I/O and MPI I/O

[10] JUQUEEN HPC System

- **The I/O node cabling connects the computing nodes via dedicated I/O nodes to storages**

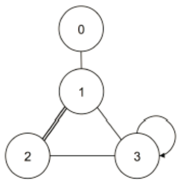
Communication Optimization by Task-Core Mappings (1)

- Approach:
  - Place often-communicating processes on neighboring nodes
  - Requires known communication behavior
  - Measurements via MPI profiling interface

Execution units → processing elements
i.e. processes → i.e. CPUs

- Identification of applicable 'task-core mapping' approach
  - E.g. graph model describes task communication & hardware characteritics
  - Obtain communication characteristics via sourcecode or profiling
  - Obtain hardware characteristics via vendor information (e.g. IBM redbooks)

- **Optimal placement of execution units to processing elements is an NP-hard-problem**
- **n! possibilities to map n execution units to the same number of n processing elements**
- **Topology aware task mapping for I/O patterns exists**



$G_t = (T, E, f)$

$T = \{t_0, t_1, t_2, t_3\}$

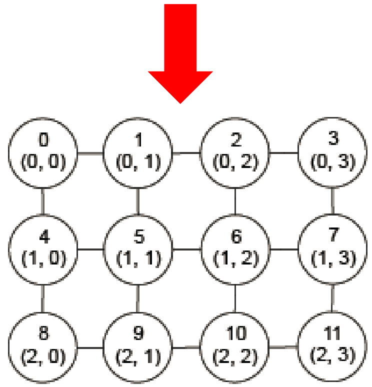
$E = \{\{t_0, t_1\}, \{t_1, t_2\}, \{t_1, t_3\}, \{t_2, t_3\}, \{t_3\}\}$

f:

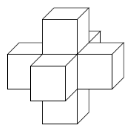
$e \in E_1$	$f_1(e)$
$\{t_0, t_1\}$	1
$\{t_1, t_2\}$	2
$\{t_1, t_3\}$	1
$\{t_2, t_3\}$	1
$\{t_3\}$	1

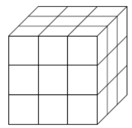
Communication Optimization by Task-Core Mappings (2)

- Application of calculated mappings
  - For regular graphs (tori): Mapping of regular shapes
  - E.g. experiments run on Bluegene/Q JUQUEEN
- Scientific application (cf. Lecture 2)
  - Heatmap as three-dimensional simulation for heat expansion
  - Values of boundary cells are exchanges with neighboring placed ranks
  - Heatmap is divided into equally sized cubes
  - Heat expansion per cube is calculated by a single rank
  - Two different expansion algorithms
  - Using e.g. 'heuristics' for task/core placements



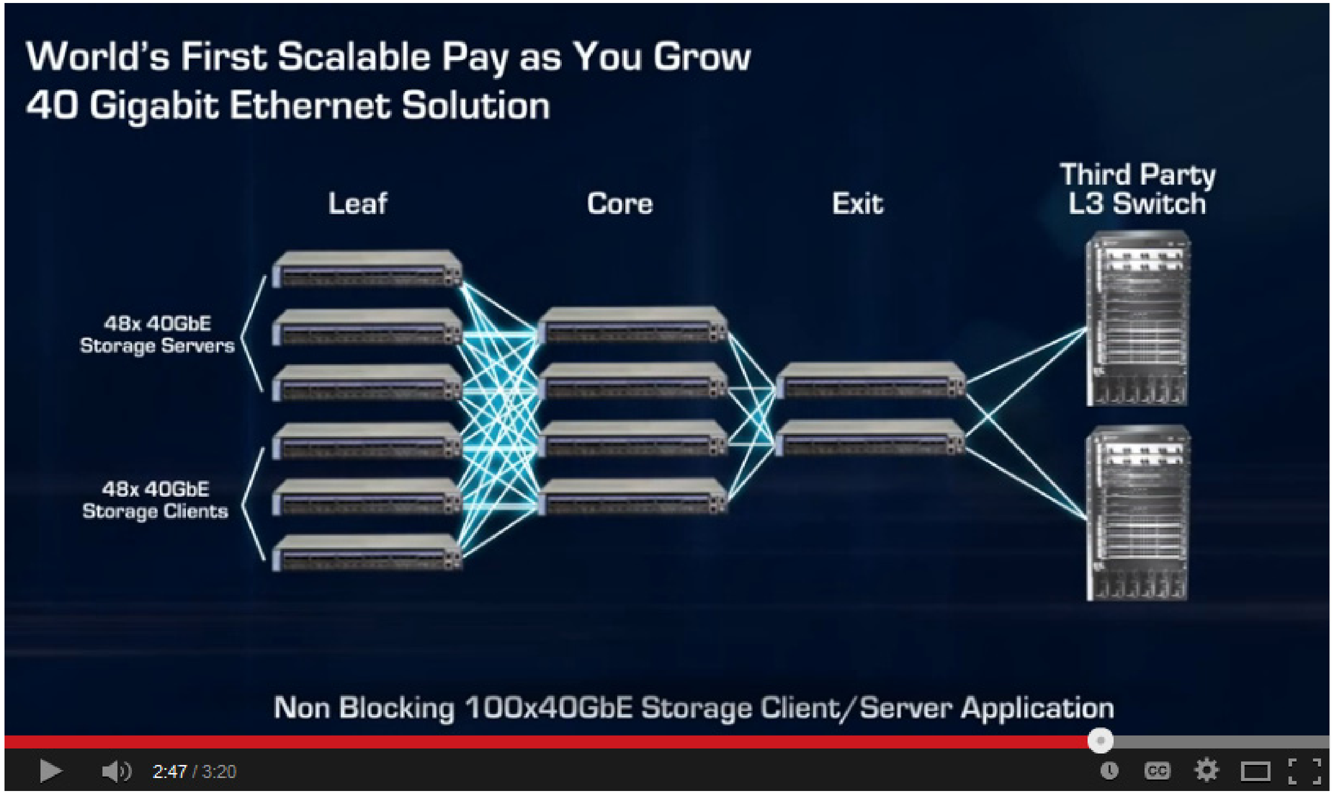
- **Optimized task core mappings enable performance gains between 1-3% (e.g., heatmap application example)**


Expansion 1


Expansion 2

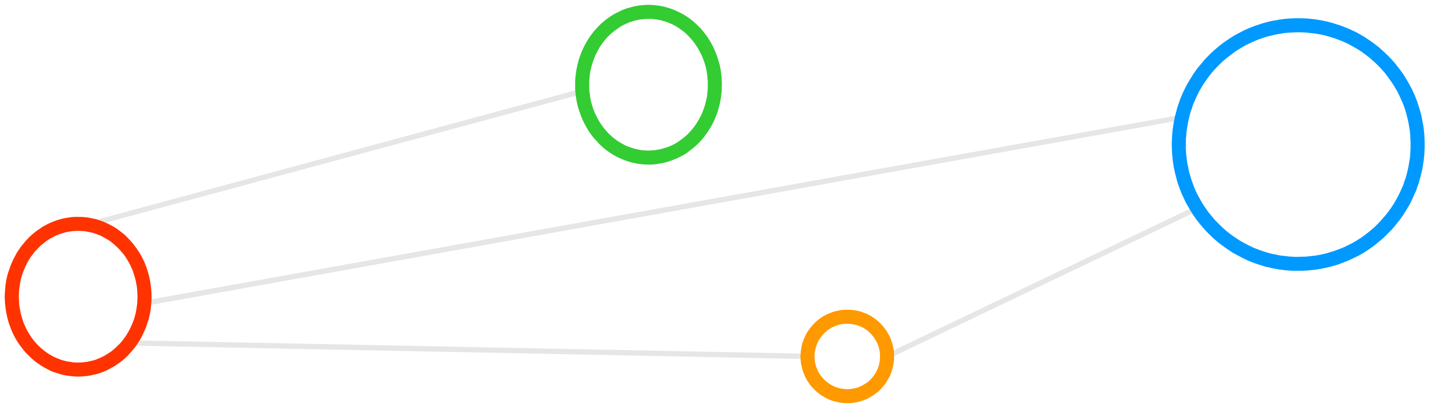
```
for (z = 1; z <= size; z++) {
    for (y = 1; y <= size; y++) {
      for (x = 1; x <= size; x++) {
        new_map[x][y][z] = ( old_map[x][y][z-1]
  + old_map[x][y-1][z] + old_map[x-1][y][z]
  + old_map[x][y][z]  + old_map[x+1][y][z]
  + old_map[x][y+1][z] + old_map[x][y][z+1] ) / 7;
      }
    }
  }
```

[Video] PEPC – Particle Acceleration Application



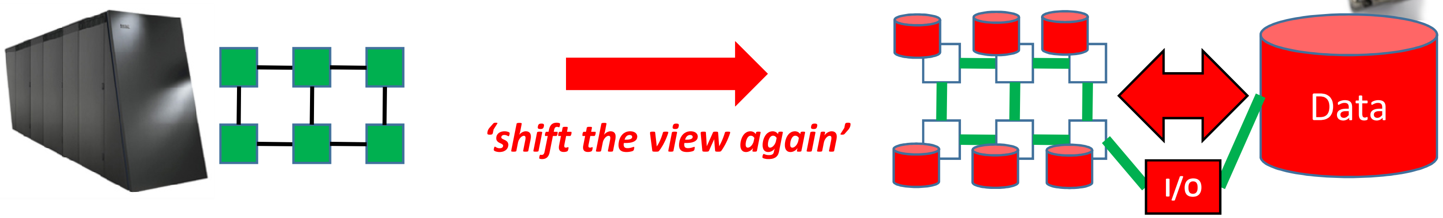
[8] Mellanox YouTube Video

MPI Parallel I/O Techniques



Parallel I/O Techniques – Motivation

- (Parallel) applications that emphasize on the importance of data
  - Not all data-intensive or data-driven applications are 'big data' (volume)
  - HPC simulations of the real world that generates very large volumes of data
- Synthesize new information from data that is maintained in distributed (partly unique) repositories and archives
  - Distributed across different organizations and computers/storages
- Data analysis applications that are 'I/O bound'
  - I/O dominates the overall execution time
  - I/O performance crucial for overall performance



> ➢ **The complementary course Cloud Computing & Big Data – Parallel & Scalable Machine & Deep Learning offers much more techniques**

What means I/O?

- Important (time-sensitive) factors within HPC environments
  - Characteristics of the computational system (e.g. dedicated I/O nodes)
  - Characteristics of the underlying filesystem (e.g. parallel file systems, etc.)

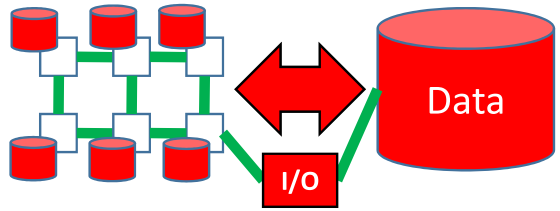
modified from [6] Introduction to High Performance Computing for Scientists and Engineers

- **Input/Output (I/O) stands for data transfer/migration from memory to disk (or vice versa) within a MPI application**

➢ **The course Cloud Computing & Big Data – Parallel & Scalable Machine & Deep Learning offers distributed file system techniques**

I/O Challenges in MPI Applications

- I/O performance bottlenecks in many 'locations in applications'
  - Understanding depends on network & I/O patterns
- During an HPC application run
  - Consider the number of processes performing I/O
  - The number of files read or written by processes
  - Take into account how the files are accessed:
    (a) serial access via one process
    (b) shared access via multiple processes
- Before/After HPC application run
  - How can necessary files be made available/archived (e.g. tertiary storage)
  - E.g. retrieving a high number of small files from tapes takes very long time



- An I/O pattern reflects the way of how a MPI application makes use of I/O (files, processes, etc.) in context of computations

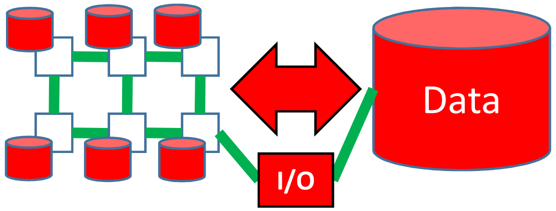
Parallel Filesystems Concept

- File Blocks
  - Distributed across multiple filesystem nodes
  - A single file is thus fully distributed across a ‘disk array’
- Advantages
  - High reading and writing speeds for a single file
  - Reason: ‘Combined bandwidth’ of the many physical drives is high
- Disadvantages:
  - Filesystem is vulnerable to disk failures (e.g. one disk fails → lose file data)
  - Prevent data loss with ‘RAID controllers’ as part of the filesystem nodes
  - Redundant Array of Inexpensive Disks (RAID) levels trade-off vs. data loss

- **A parallel file system is optimized to specifically support concurrent file access**
- **One file that is written to a parallel filesystem is broken up into ‘blocks’ of a configured size (e.g. typically less than 1MB each)**
- **Prevent data loss with Redundant Array of Inexpensive Disks (RAID) levels**

➢ **The course Cloud Computing & Big Data – Parallel & Scalable Machine & Deep Learning offers data storage details (e.g. RAID levels)**

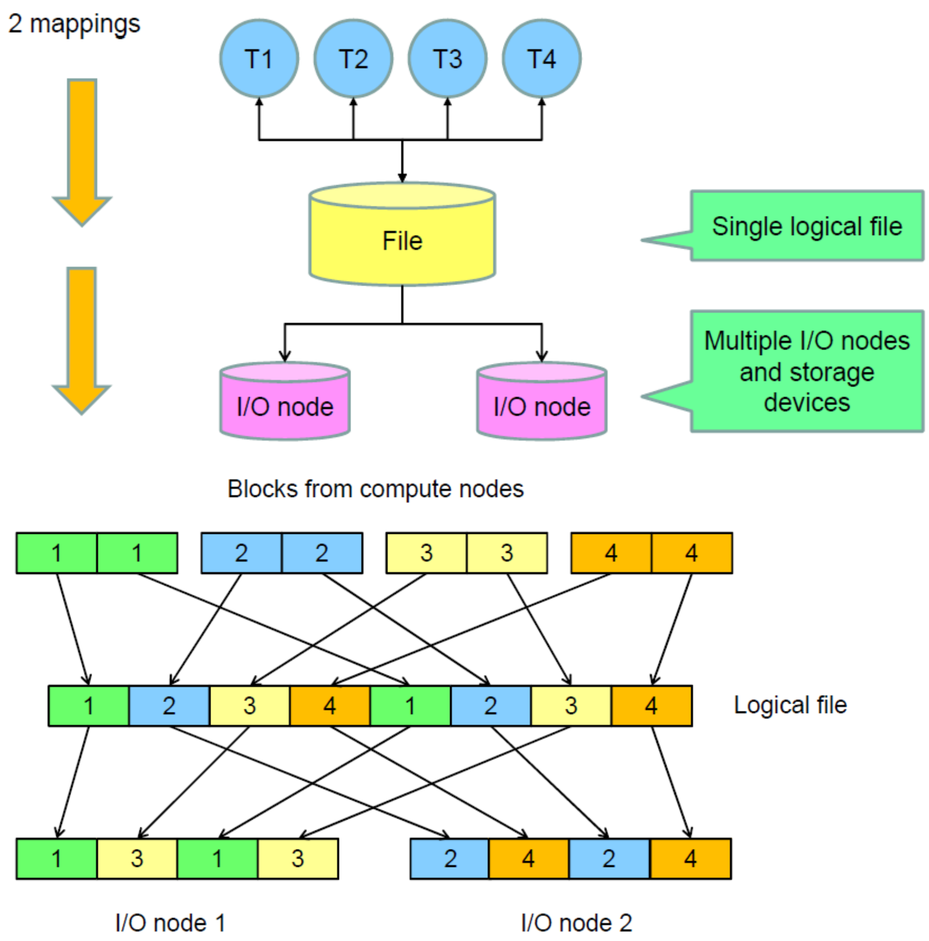
Examples of Parallel File Systems



- General Parallel File System (GPFS) / IBM Spectrum Scale
  - Developed by IBM
  - Available for AIX and Linux
  - Quite expensive solution (but powerful)
  - Moved from HPC-centric computing to 'Big Data' solution (in sales & marketing division)
- Lustre
  - Developed by Cluster File Systems, Inc. (bought by Sun)
  - Movement towards 'OpenLustre'
  - Name is amalgam of 'Linux and clusters'
  - As it is free software it becomes more and more used today
- Parallel Virtual File System (PVFS)
  - Platform for I/O research and production file system for cluster of workstations ('Beowulfs')
  - Developed by Clemson University and Argonne National Laboratory

> - **Widely used parallel file systems are General Parallel File System (GPFS) that is a commercial solution from IBM and Lustre that is open source**
> - **In 2015 IBM rebranded GPFS as IBM Spectrum Scale due to 'Big Data' customers and became a central solution for data-intensive sciences & artificial intelligence**

Concurrent File Access & Two Level Mapping



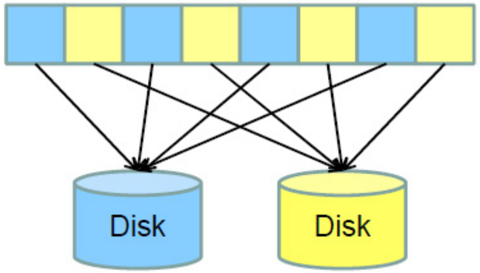
- **Concurrent file access means that multiple processes can access the same file at the same time**
- **Parallel file systems handle concurrent file access via 'single logical files' over multiple I/O nodes**
- **A two Level Mapping means to distribute blocks from compute nodes via logical files (1st level) using underlying multiple I/O nodes (2nd level)**

General Striping Technique

- Striping technique transforms view from a file to flexible ‘blocks‘

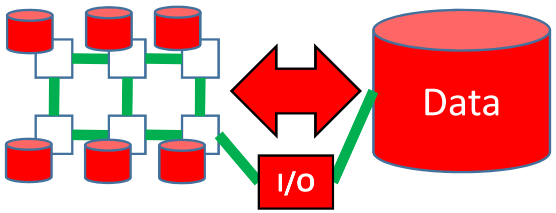
> - **Striping refers to a technique where one file is split into fixed-sized blocks that are written to separate disks in order to facilitate parallel access**

- Striping is a general technique that appears in different contexts
  - Many fields in computer science make use of striping (e.g., data transfer too)
- Two major important factors (to be configured)
  - (e.g. used in MPI I/O ‘hints’ also → later in this lecture)
  - ‘Striping factor’: number of disks
  - ‘Striping unit’: block size
  - Bit-level vs. block-level striping

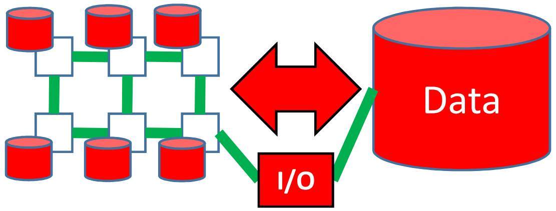


Parallel File Access

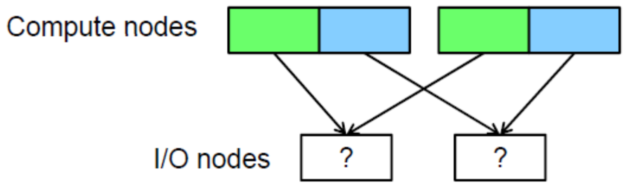
- Comparison with ‘sequential file system’ increases understanding
  - File system translates ‘file name’ into a File Control Block (FCB)
- Parallel File Systems
  - Every ‘I/O node’ manages a subset of the blocks
  - Consequence: Every file has (better: needs) an FCB on every I/O node
- File Access: Two ways to locate FCBs for a file
  - Every I/O node maintains directory structure
  - Central name server: Avoids replication of directory data
- File Creation
  - Filesystem chooses ‘the first’ I/O node (varies)
  - This particular I/O node (‘base node’) will store the first block of the file
  - Specific block is located when first I/O node and ‘striping pattern’ is known
- Question: What about ‘sequential consistency’ when writing?



Sequential Consistency

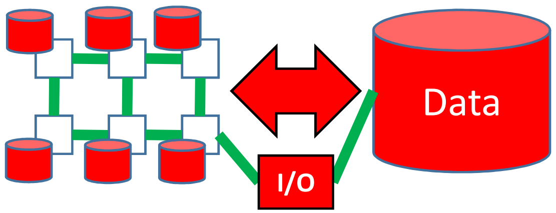


- Two processes on different compute nodes
  - Assumption: Both write to the 'same range of locations in a file'
- 'Sequential consistency'
  - Requires that all I/O nodes write their portions in the same order
  - Write request should appear to occur in well-defined sequence
  - But hard to enforce – I/O nodes may act independently



- Selected Possible Solutions
  - Locking entire files - Prevents parallel access (not an option)
  - Relaxed consistency semantics – application developer is responsible
  - Locking file partitions – prevents access to certain file partitions

File Pointers

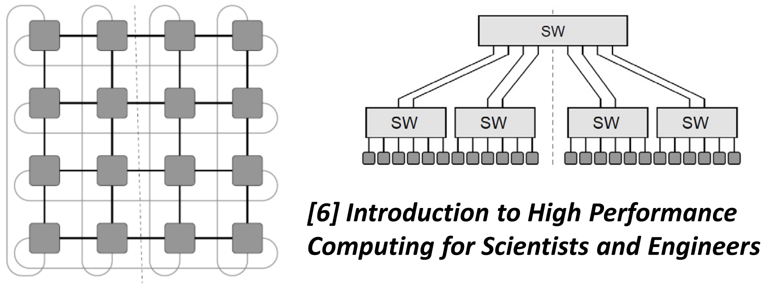


- MPI Applications
  - Need to be aware of 'which processes use which parts of the file'
  - May require processes to skip file sections 'owned by others'
- Shared File Pointers
  - Common in shared-memory programs
  - Inefficient – serializes requests (update file pointer before completing request, 'eager update')
  - Inconsistencies if seek and write operations are separated
- Improvements of Usage
  - Better use 'separate file pointers' or atomic seek & write
  - In UNIX `pread()` and `pwrite()` allow specification of 'explicit offset'

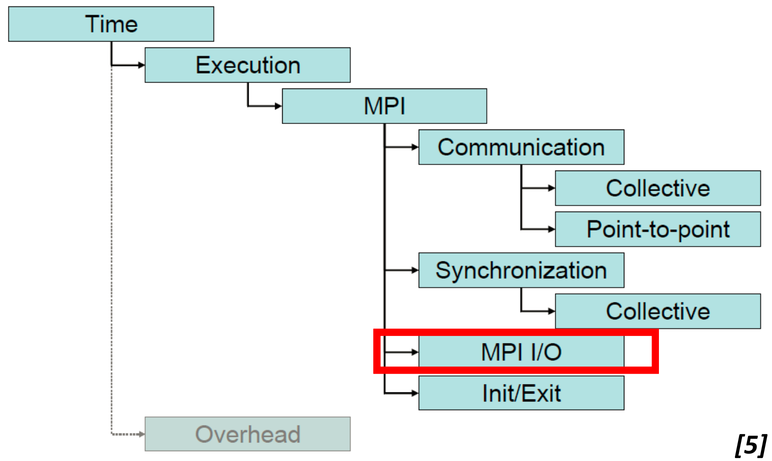
```
Thread 1:                      Thread 2:
seek(location = 100);          seek(location = 200);
write(..., length = 50);       write(..., length = 150);
```

Optimization & Dependencies on Hardware & I/O – Revisited

- Optimizations in terms of software & hardware are important
  - Optimization can be interpreted as using ‘dedicated‘ hardware features (if available)
  - E.g. network interconnections enable different used ‘network topologies‘ (varies in different systems)
  - E.g. parallel codes are tuned applying parallel I/O with parallel filesystems (if parallel filesystem exists)



[6] Introduction to High Performance Computing for Scientists and Engineers

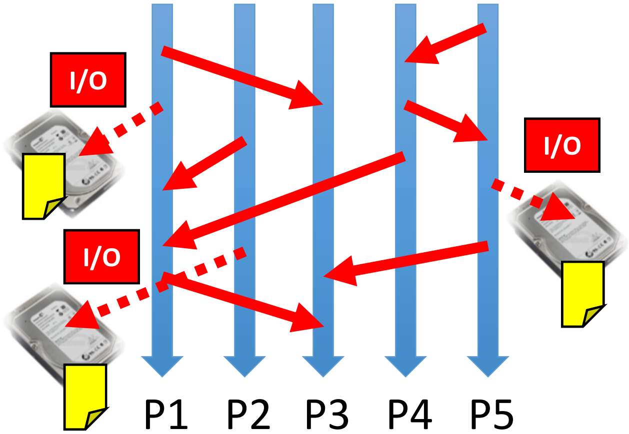


[5] Metrics tour

> **Lecture 9 on debugging, profiling & performance toolsets offers insights into performance analysis tools to understand MPI code better**

MPI I/O

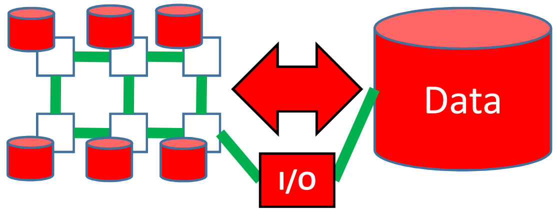
- Different operation modes
  - 'Blocking mode' to finish data operations, then continue computations
  - 'Non-blocking mode' (aka asynchronously) to perform computations while a file is being read or written in the background (typically more difficult to use)
- Supports the concept of 'collective operations'
  - Processes can access files each on its own or all together at the same time
- Provides advanced concepts
  - E.g., file views & data types/structures

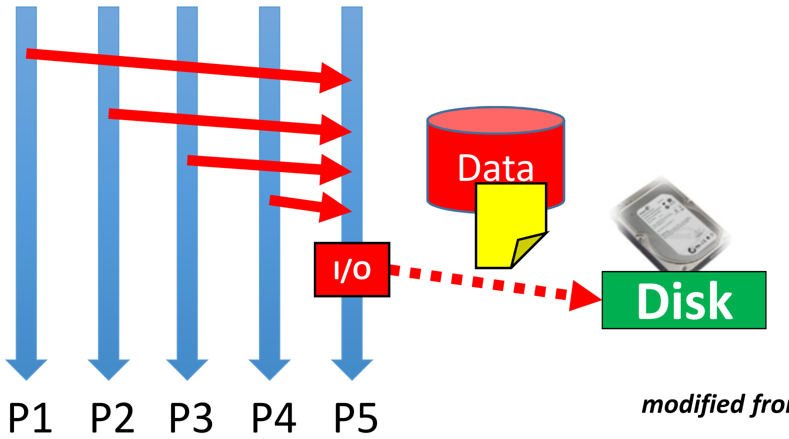


- **MPI I/O provides 'parallel I/O' support for parallel MPI applications**
- **Writing/Receiving files is similar to send/receive MPI messages, but to disk**

Serial I/O: One Process on behalf of Many Processes

- Only one process performs I/O on behalf of all other processors
  - Data aggregation or duplication
  - Limited by single I/O process (e.g. determined by rank as writer role)
- No scalability for (big) data-intensive computing
  - Time increases linearly with amount of (big) data
  - Time increases with number of processes of the parallel application



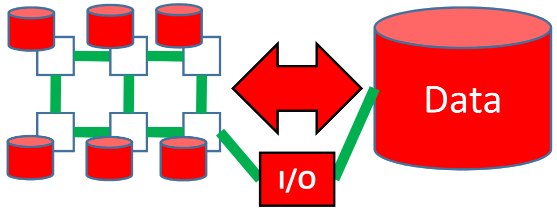


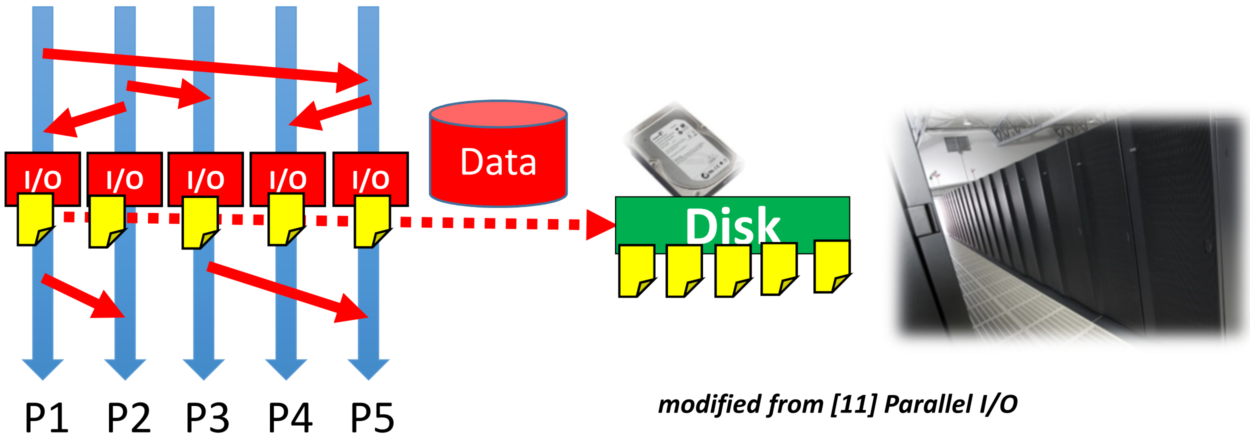
modified from [11] Parallel I/O

- Serial I/O: One process on behalf of many means that one process takes care of all I/O tasks
- Serial I/O increases communication and is slow as well as including load imbalance risks

Parallel I/O: One file per Process

- All processors perform I/O to individual files
  - Limited by file system capabilities
- No scalability for large number of processors
  - Number of files creates bottleneck with metadata operations
  - Number of simultaneous disk accesses creates 'contention' for file system resources
  - E.g., the disk cannot keep up with file I/O requests



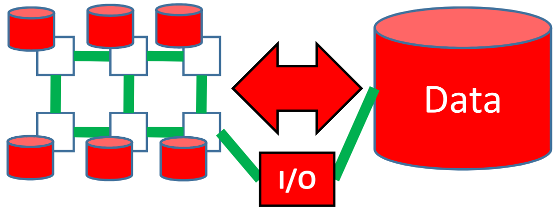


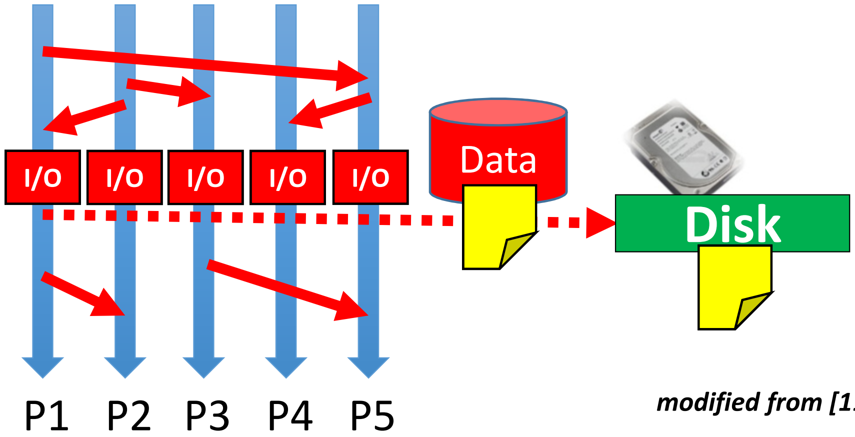
modified from [11] Parallel I/O

- **Parallel I/O: One file per process means that each process takes care of local I/O tasks alone**
- **Parallel I/O is good for scratch but not for output files in applications despite I/O balance**

Parallel I/O: Shared File

- Each process performs I/O to a single file
  - The file access is ‘shared’ across all processors involved
  - E.g. MPI/IO functions represent ‘collective operations‘
- Scalability and Performance
  - ‘Data layout’ within the shared file is crucial to the performance
  - High number of processors can still create ‘contention’ for file systems





modified from [11] Parallel I/O

- Parallel I/O: shared file means that processes can access their ‘own portion’ of a single file
- Parallel I/O with a shared file like MPI/IO is a scalable and even standardized solution

Collective MPI-I/O: Writing integers to a file example

```
#include <stdio.h>
#include <mpi.h>

int main (int argc, char** argv) {
  int rank, size;

  MPI_File fh;
  MPI_Info info;

  char *file_name = "outputfile";

  int buf[10];

  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  printf("Hello World, I am %d of %d\n", rank, size);

  MPI_Info_create(&info);

  int rc = MPI_File_open( MPI_COMM_WORLD, file_name,
                          MPI_MODE_CREATE | MPI_MODE_RDWR,
                          info, &fh);
  buf[0] = rank;

  // MPI_File_write_ordered(fh, buf, 1, MPI_INT, &status);
  MPI_File_write(fh, buf, 1, MPI_INT, &status);

  rc = MPI_File_close(&fh);

  MPI_Finalize();
  return 0;
}
```

- A MPI_File represents a file handler that reprents the file and process group of a communicator
- A MPI_Info represents a list of key/value pairs used for providing information to MPI-I/O

- Specifying a file_name that should be opened (or even be created) – but attention: The format is highly implementation dependent

- MPI_Info_create creates an MPI_Info object to be used to provide information to MPI-I/O

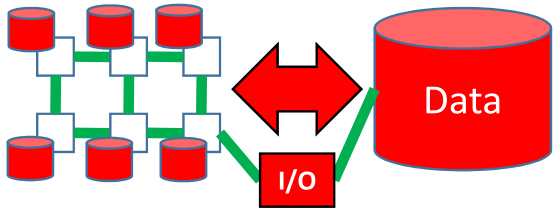
- MPI_File_open opens a specific file collectively across all specified processes being part of the used communicator and sets a file handle

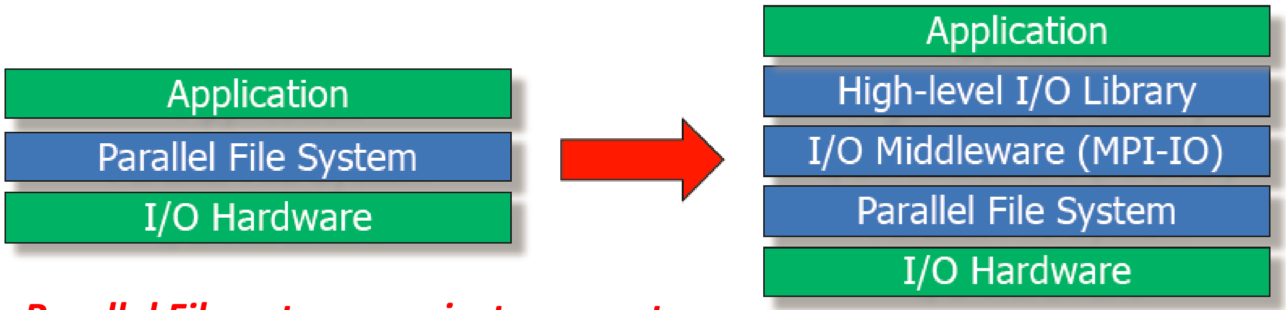
- Requires a buffer (here integer array) of a certain size (e.g. buf[10])
- Requires values for the buffer: here the rank of each MPI process that might be used as identification for further values following in the next parts of the corresponding file is used
- MPI_File_write or related versions write the <u>binary output</u> to the file
- Different between MPI_File_write() and MPI_File_write_ordered() is that the out is not ordered according to ranks or ordered by ranks

- MPI_File_close closes a specific file identified via a certain file handle

MPI I/O & Parallel Filesystems

- Understanding and tuning parallel I/O is needed with 'big data'
  - Leverage aggregate communication and I/O bandwidth of client machines
- Support: Add additional software components/libraries layers
  - Coordination of file access & mapping of application model to I/O model
  - Components and libraries get increasingly specialized / layer
  - High-Level I/O libraries like NetCDF or Hierarchical Data Format (HDF) are standards in the community



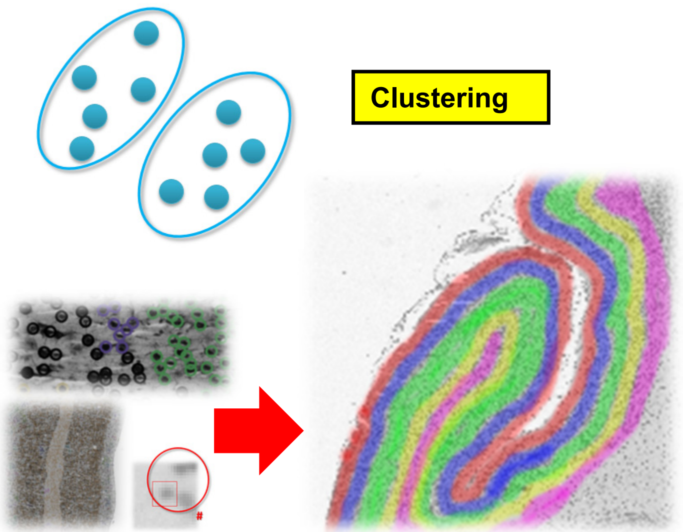


Parallel Filesystems are just one part out of three in the whole I/O process

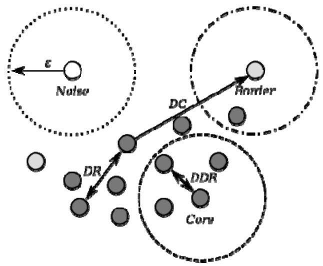
[9] R. Thakur, PRACE Training, Parallel I/O and MPI I/O

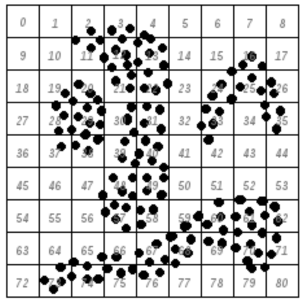
➢ **Lecture 5 offers more details on using Parallel I/O and portable data formats in various simulation sciences & data science applications**

Data Science Example: Using High-Level I/O Hierarchical Data Format (HDF)

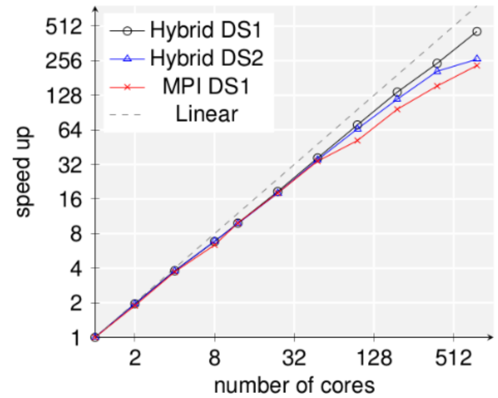


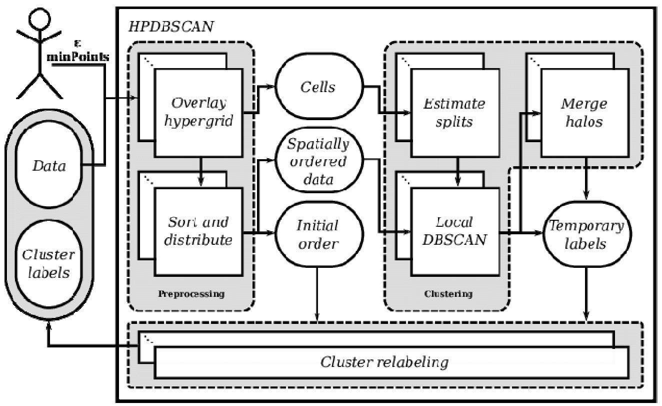
[13] M. Goetz and M. Riedel et al,
Proceedings IEEE Supercomputing Conference, 2015

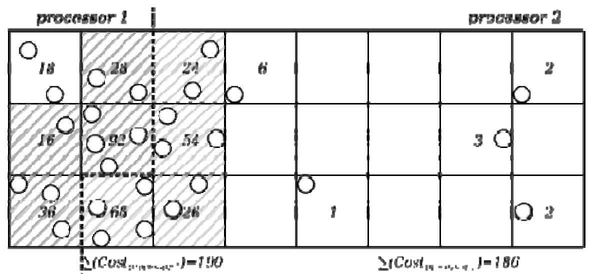




Overlayed spatial grid

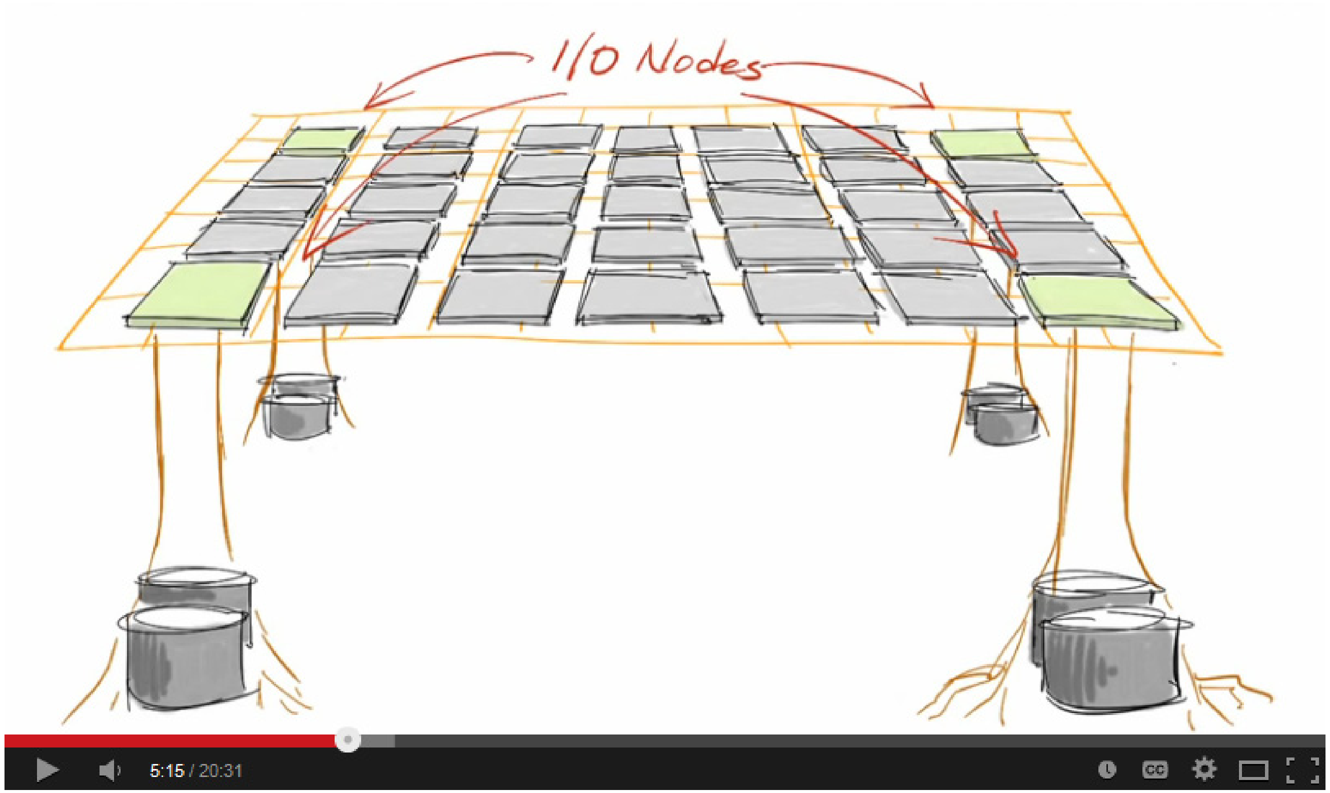






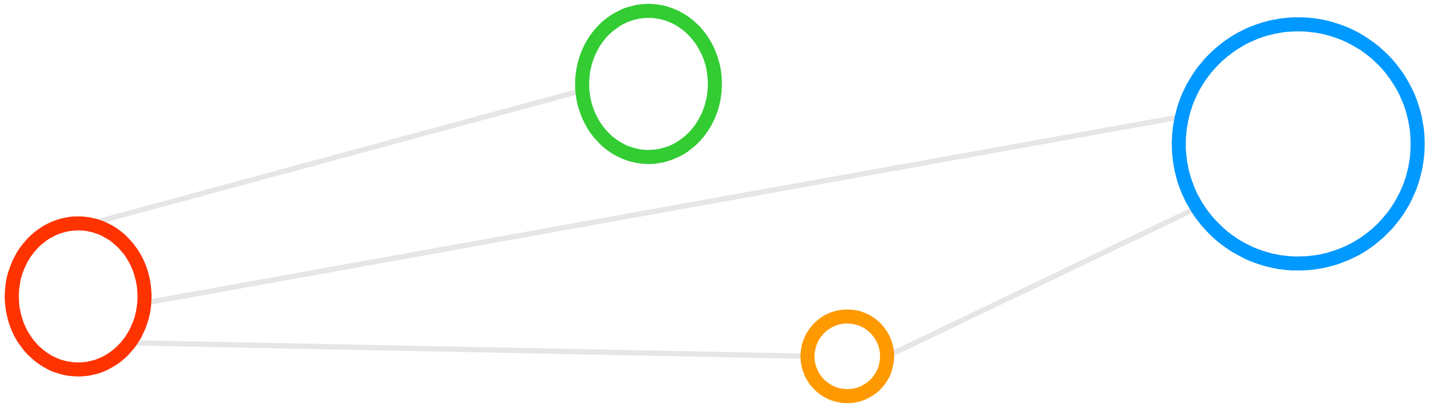
- Lecture 5 offers more details on using Parallel I/O and portable data formats in various simulation sciences & data science applications

[Video] Parallel I/O with I/O Nodes



[12] YouTube Video, 'Simplifying HPC Architectures'

Lecture Bibliography



Lecture Bibliography (1)

- [1] LLNL MPI Tutorial, Online:
  https://computing.llnl.gov/tutorials/mpi/
- [2] Introduction to Groups and Communicators, Online:
  http://mpitutorial.com/tutorials/introduction-to-groups-and-communicators/
- [3] German Lecture 'Umfang von MPI 1.2 und MPI 2.0'
- [4] The MPI Standard, Online:
  http://www.mpi-forum.org/docs/
- [5] M. Geimer et al., 'SCALASCA performance properties: The metrics tour'
- [6] Introduction to High Performance Computing for Scientists and Engineers, Georg Hager & Gerhard Wellein, Chapman & Hall/CRC Computational Science
- [7] Wolfgang Frings, 'HPC I/O Best Practices at JSC', Online:
  http://www.fz-juelich.de/ias/jsc/DE/Leistungen/Dienstleistungen/Dokumentation/Praesentationen/folien-parallelio-2014_table.html?nn=469624
- [8] YouTube Video, 'Mellanox 10 and 40 Gigabit Ethernet Switch Family', Online:
  http://www.youtube.com/watch?v=o9BLItx2vDg
- [9] Rajeev Thakur, Parallel I/O and MPI-IO, Online:
  http://www.training.prace-ri.eu/uploads/tx_pracetmo/pio1.pdf
- [10] JUQUEEN, Online:
  http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/JUQUEEN_node.html
- [11] Parallel I/O, YouTube Video, Online:
  http://www.youtube.com/watch?v=cXbEVsExU9c
- [12] Big Ideas: Simplifying High Performance Computing Architectures, Online:
  https://www.youtube.com/watch?v=ISS_OGVamBk

Lecture Bibliography (2)

- [13] M. Goetz, C. Bodenstein, M. Riedel, ‘HPDBSCAN – Highly Parallel DBSCAN’, in proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC2015), Machine Learning in HPC Environments (MLHPC) Workshop, 2015, Online: https://www.researchgate.net/publication/301463871_HPDBSCAN_highly_parallel_DBSCAN


services
measurement
within
fusion
new
funding
Policy-based
concepts
forms
cross-disciplinary
resources
device
Analysis
Centre
climate
Computer
expertise
Approaches
construction
Fusion
year
like
Computational
Test
E.g
Computing
disciplines
results
Enable
different
Cross-Disciplinary
High
big
nodes
PB
community
Environment
Modelling
EU
methods
centers
Key
scientific
important
Science
etc
DKRZ
Use
storage
basis
Infrastructure
files
model
international
SMAQ
engineering
References
formats
simulations
ENES
technologies
brain
increasing
via
EUDAT
physical
IPCC
Big
Project
Data
MapReduce
create
processing
Common
TB
Methods
computing
Using
European
databases
codes
GB
Supercomputing
Work
images
analysis
Large
Summary
data-intensive
Service
Scientific
levels
often
use
DLCL
Level
stored
Resources
analyze
external
JSC
computational
Research
Matters
Concrete
performance
research
e.g
science
HPC
large-scale
Juelich
system
datasets
acquisition
compute
Landscape
System
Climate
modelling
AR5
Hardware
cores
access
hundreds
Nutshell
Services
management
DLCLs
Understanding
structures
Technologies
Earth
distributed
Structure
current
network
Germany
Simulation
directory
project
General
Provide
NASA
Energy
systems
day
Processing
Health